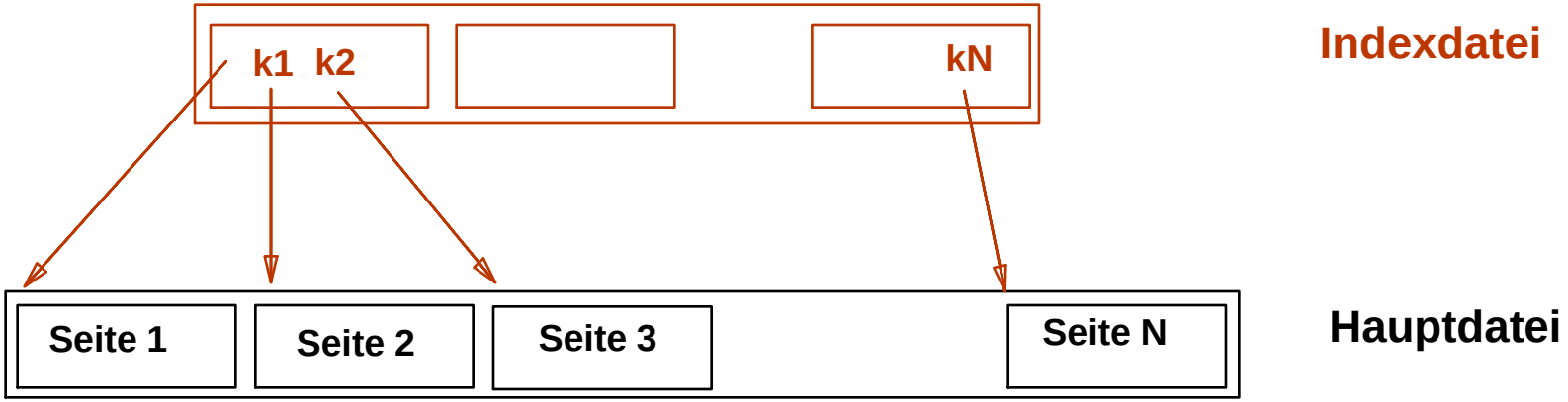
Baum-Indexverfahren

Einführung

- Drei Alternativen, wie Dateneinträge **k*** im Index aussehen können:
  1. Datensatz mit Schlüsselwert **k**
  2. <**k**, ID des Datensatzes mit dem Wert des Suchschlüssels **k**>
  3. <**k**, Liste von Datensatz-IDs mit Suchschlüssel **k**>
- Auswahl einer Alternative ist orthogonal zur Indexierungstechnik, die genutzt wird, um Dateneinträge mit einem gegebenen Schlüsselwert **k** zu plazieren.
- Index mit Baumstruktur unterstützt Bereichsanfragen (Range Query) und Lookups mit einzelnen Schlüsselwerten
- ISAM: statische Struktur (indexsequentieller Zugriff); B+ Baum: dynamisch, sehr anpassungsfähig beim Einfügen und Löschen

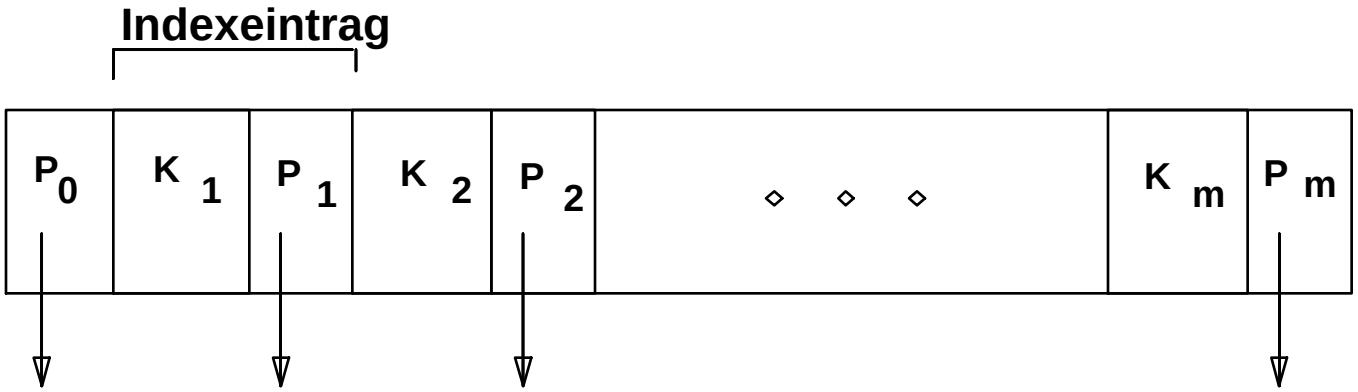
Bereichs-Anfragen

- *“Finde alle Studenten mit Durchschnitt > 3.0”*
  - Wenn Daten in sortierter Datei, dann mache binäre Suche, um den ersten solchen Studenten zu finden, dann lese sequentiell weiter, um die anderen zu finden (scan).
  - Kosten einer Binärsuche können ziemlich hoch sein
- Einfache Idee: Anlegen einer Indexdatei

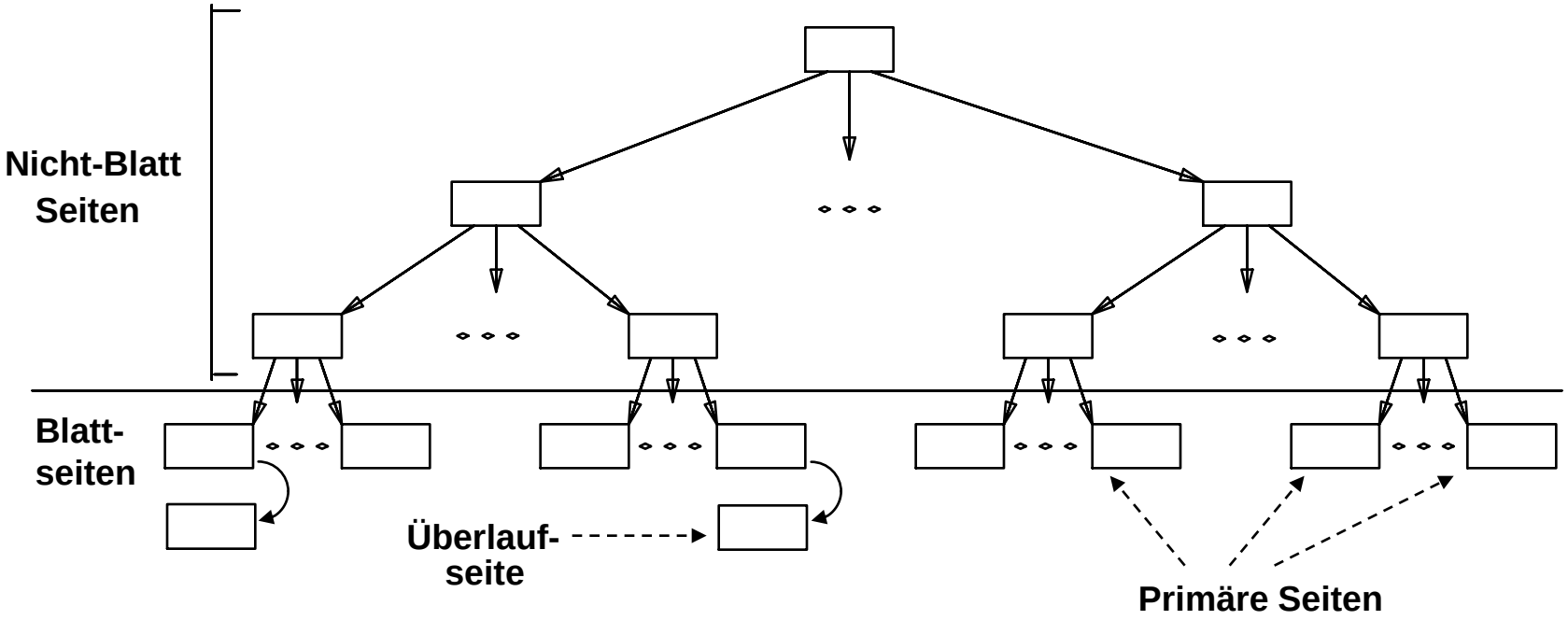


Binärsuche kann somit auf (kleinerer) Indexdatei ausgeführt werden

ISAM



Indexdatei kann immer noch sehr groß sein. Aber die Index-Idee kann mehrfach angewandt werden (mehrstufiger Index)!



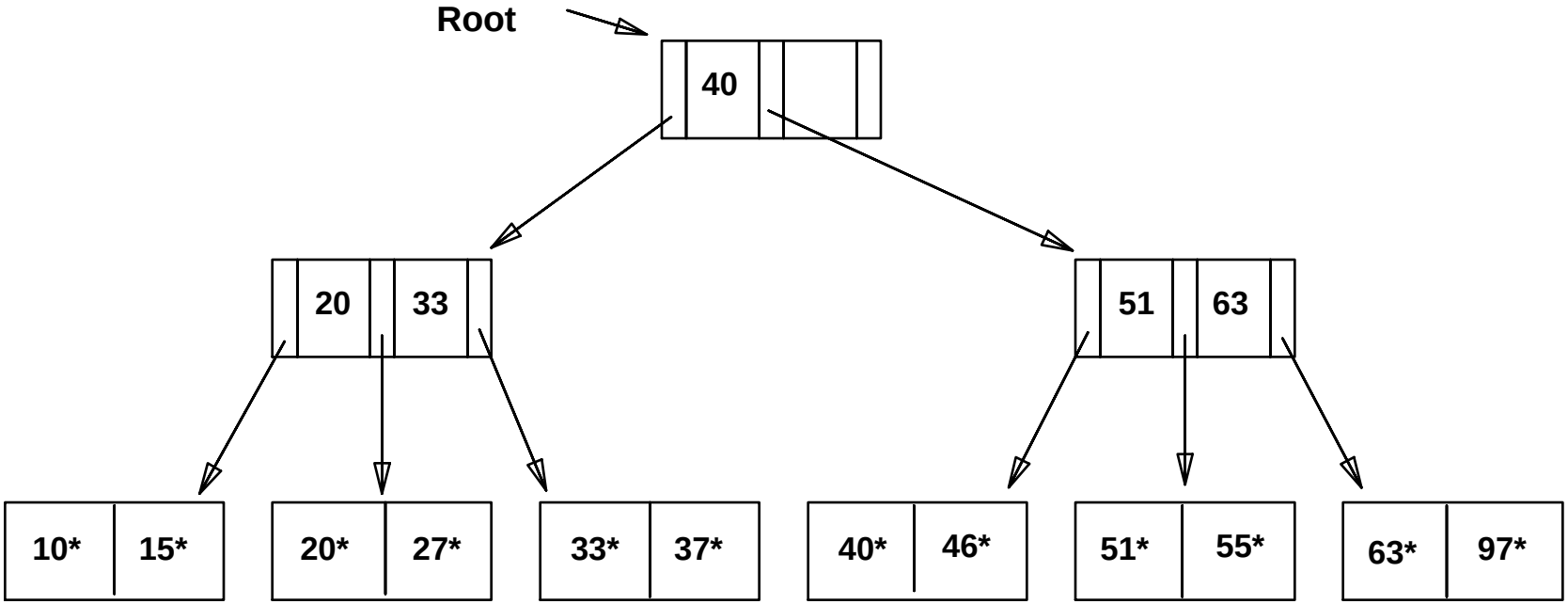
Blattseiten enthalten Dateneinträge.

Mehr über ISAM

- *Erzeugung einer Datei*: Blattseiten (Datenseiten) werden sequentiell allokiert, sortiert nach dem Suchschlüssel; dann werden Indexseiten allokiert, dann Speicher für Überlaufseiten.
- *Indexeinträge*: <Suchschlüssel-Wert, Seiten-ID>; sie lenken die Suche nach Dateneinträgen, die sich in den Blattseiten befinden.
- *Search*: Beginne bei der Wurzel; mache Vergleich mit Schlüsselwert auf dem Weg zu einem Blatt. Kosten $\log_F N$; F = # Einträge/Indexseite, N = # Anzahl Blattseiten
- *Insert*: Finde das Blatt, wohin der Dateneintrag gehört und füge ihn dort ein.
- *Delete*: Finde das Blatt und lösche Eintrag von dort; wenn die Überlauf-Seite leer ist, deallokiere sie (d.h. gebe Speicher frei)
- Statische Baum-Struktur
  - Insert / Delete betrifft nur die Blattseiten

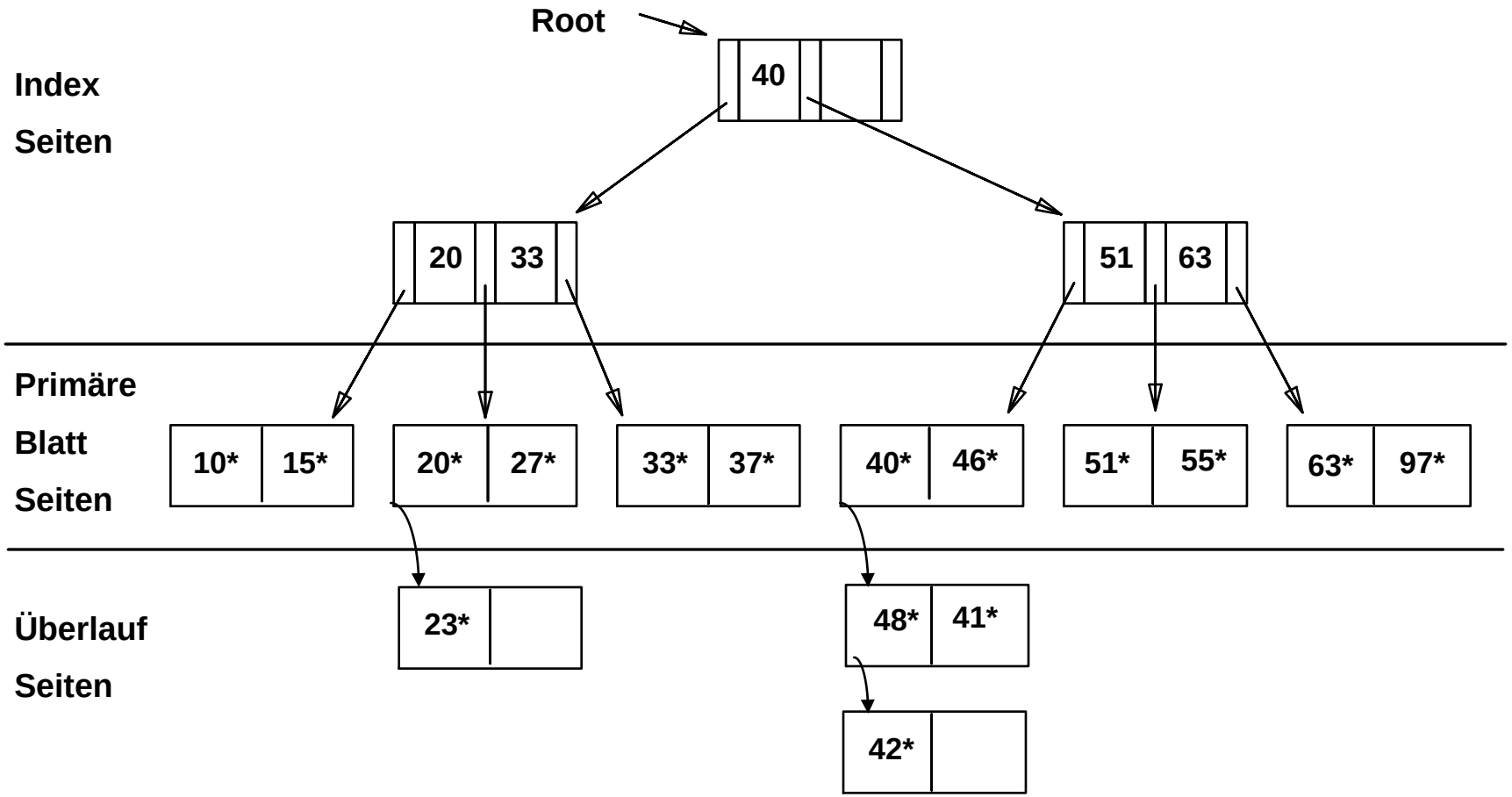
Beispiel ISAM-Baum

- Jeder Knoten kann 2 Einträge aufnehmen; kein Bedarf für Pointer zur nächsten Seite



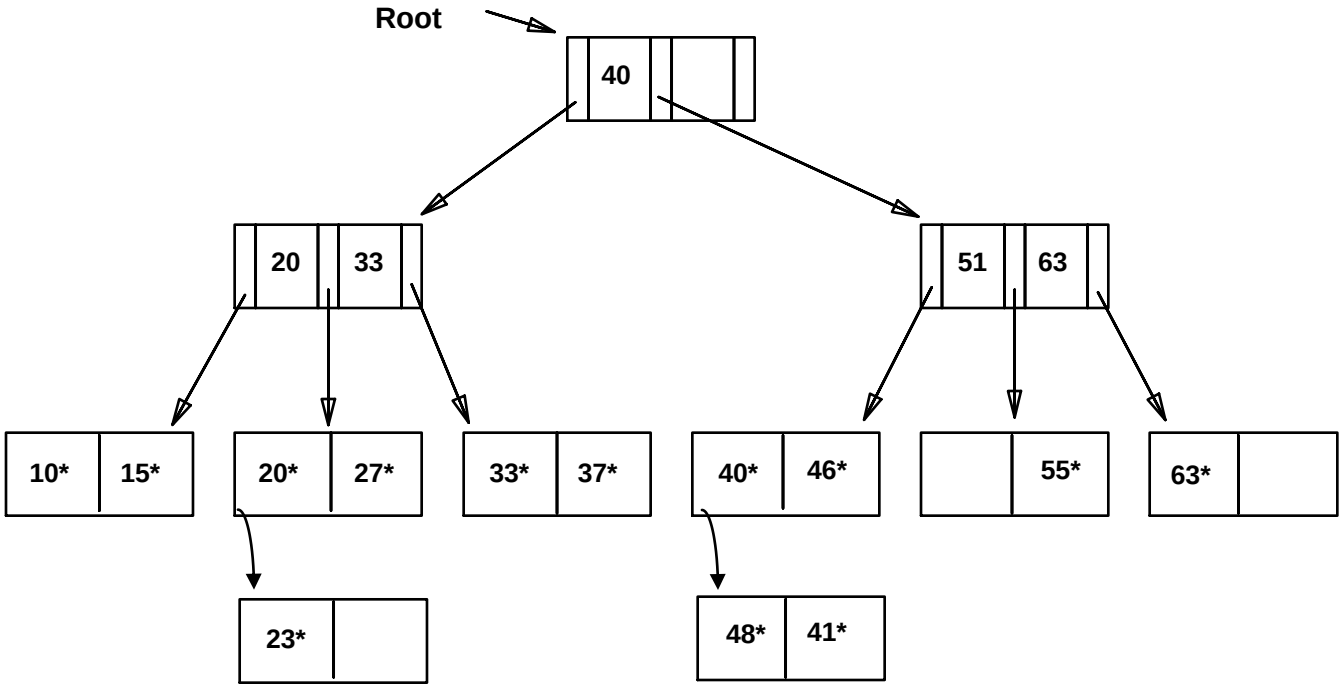
Einfügen

Insert 23*, 48*, 41*, 42* ...



Löschen

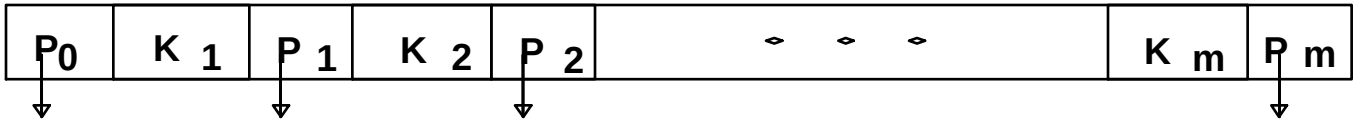
... Delete 42*, 51*, 97*



Beachte: 51 erscheint nur in Indexstufen, aber nicht im Blatt!*

B+ Baum: Meistgenutzter Index

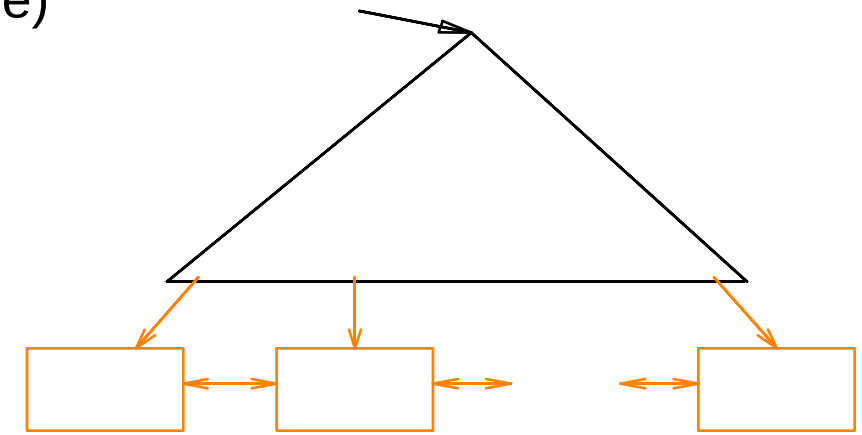
- Ordnung **d** des Baumes: Jeder Zwischenknoten enthält $\mathbf{d} \leq m \leq 2\mathbf{d}$ Einträge und somit mindestens d+1 und höchstens 2d+1 Söhne
- Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne
- Höhe h des Baumes: Länge des Pfades von der Wurzel bis zu einem Blatt (im allg. 3-4)
- Format eines Knotens:



- Pi verweist auf einen Teilbaum, in dem für alle Schlüsselwerte K gilt: $K_i \leq K < K_{i+1}$
- Spezialfall: P_0 verweist auf einen Baum, in dem alle $K < K_1$
  P_m verweist auf einen Baum, in dem alle $K \geq P_m$
- Insert / Delete erhalten den Baum höhenbalanciert
- Mindestens 50% Belegung garantiert
- Suchkosten: $\log_d N$ (d=Ordnung, N=Anzahl der indizierten Datensätze)

B-Bäume in der Praxis

- Typische Ordnung: 100. Typischer Füllfaktor: 67%.
  - durchschnittliche Anzahl von Söhnen = 133
- Typische Kapazitäten:
  - Höhe 4: 133^4 = 312,900,700 Sätze
  - Höhe 3: 133^3 = 2,352,637 Sätze
- Die obersten Stufen können im Puffer gehalten werden:
  - Level 1 = 1 Seite = 8 Kbytes
  - Level 2 = 133 Seiten = 1 Mbyte
  - Level 3 = 17,689 Seiten = 133 Mbytes
- Dateneinträge nur in Blattknoten (B+ Bäume heißen auch B* Bäume)

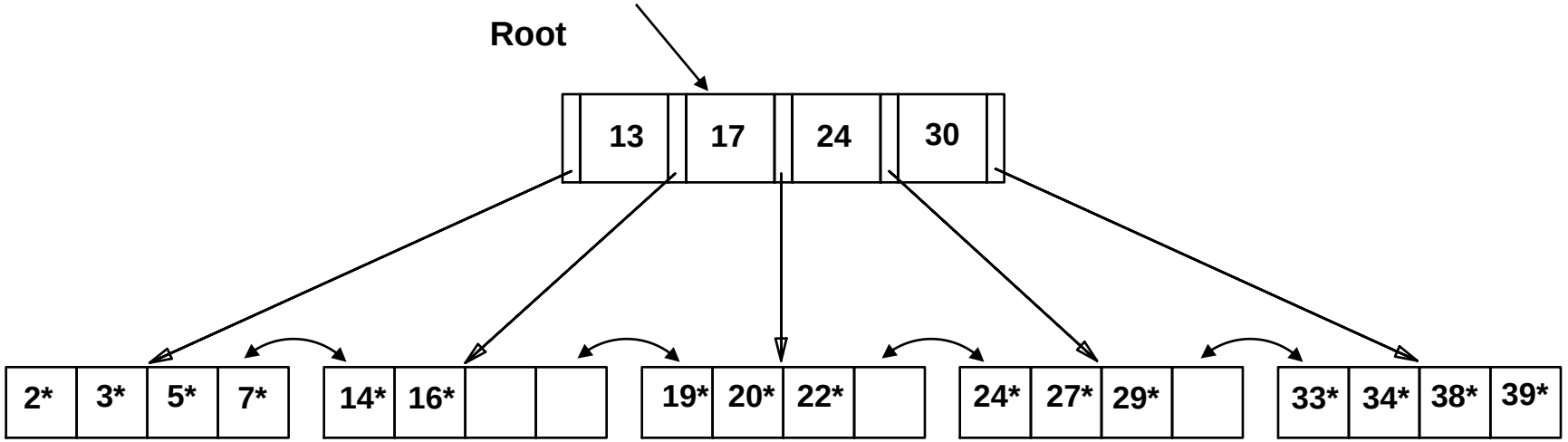


Index-Einträge
(Direkte Suche)

Dateneinträge
("Sequenz")

Beispiel B+ Baum

- Suche beginnt an der Wurzel, durch Vergleiche mit dem Schlüsselwert bis zu einem Blatt (wie in ISAM).
- Suche nach 5*, 15*, allen Dateneinträgen >= 24* ...



Wir finden 15 nicht auf dem entsprechenden Blatt, somit nicht im Baum!*

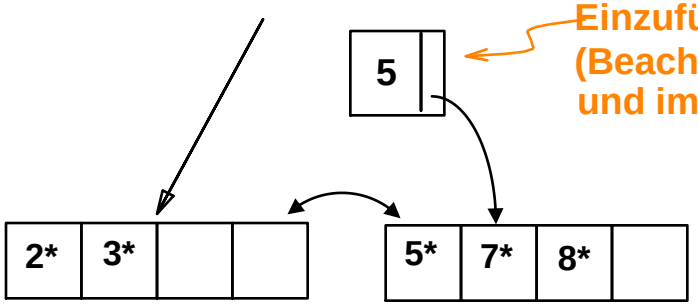
Einfügen eines Eintrags in einen B+ Baum

- Finde das richtige Blatt *L.*
- Lege den Eintrag in *L* ab.
  - Wenn in *L* genug Platz, *fertig!*
  - Sonst, *splitte* *L (*in *L* und einen neuen Knoten *L2)*
    - Umverteilung der Einträge gleichmäßig, **kopiere** den mittleren Schlüssel nach oben
    - Füge einen Indexeintrag ein beim Vater von *L*, der auf *L2* verweist
- Dies kann rekursiv geschehen
  - Um einen Index-Knoten zu splitten, verteile die Einträge gleichmäßig, aber **ziehe** den mittleren Schlüssel hoch. (Anders als beim Splitten von Blättern!)
- Durch Splitten "wächst" der Baum; ein Split der Wurzel erhöht die Baumhöhe.
  - Baumwachstum heißt: wird *breiter* oder *eine Stufe höher an der Spitze.*

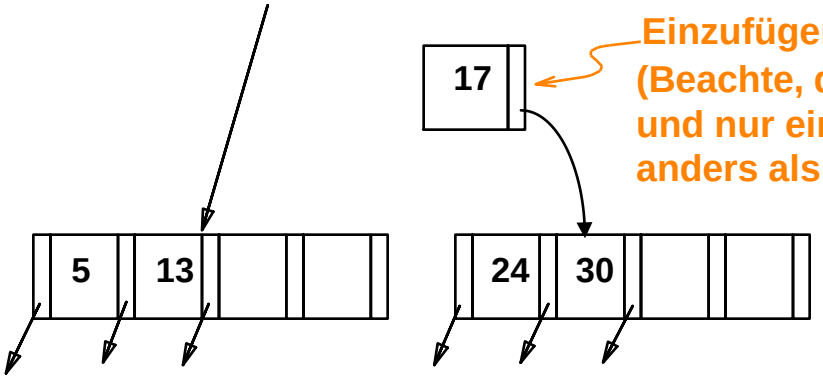
Einfügen in den Beispiel-Baum

Insert 8*

- Beachte, wie die minimale Belegung beim Splitting von Blatt und Indexseite garantiert wird.
- Beachte die Differenz zwischen *Hochkopieren* und *Hochziehen*; (hat einen Grund)!



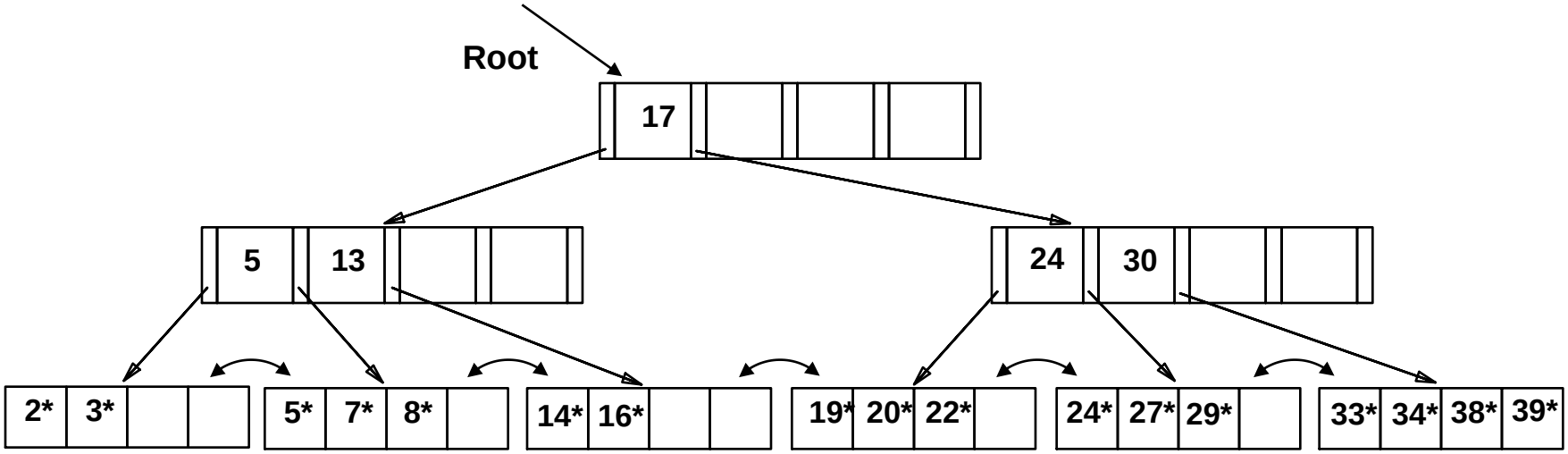
Einzufügender Eintrag im Vaterknoten. (Beachte, daß 5 hochkopiert wird und im Blatt fortbestehen bleibt.)



Einzufügender Eintrag im Vaterknoten (Beachte, daß 17 hochgezogen wird und nur einmal im Index erscheint - anders als beim Split eines Blattes)

Beispiel-Baum nach dem Einfügen

Insert 8*



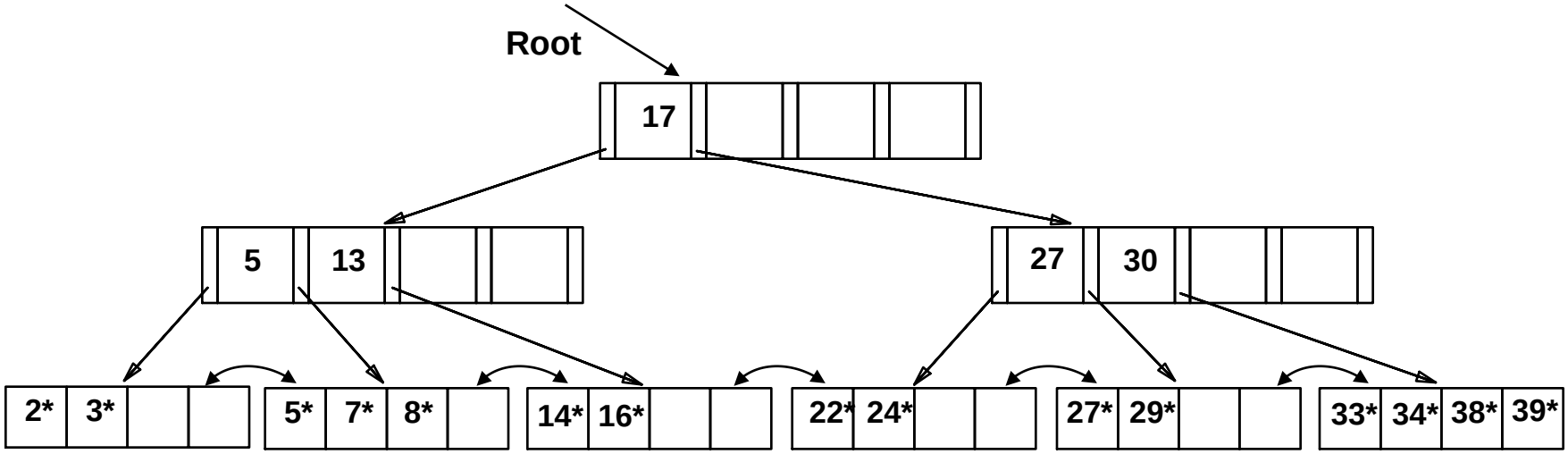
- Beachte, daß die Wurzel gesplittet wurde, somit wird die Höhe des Baums größer.
- In diesem Beispiel könnten wir einen Split durch eine Umverteilung der Einträge vermeiden, wird gewöhnlich in der Praxis nicht gemacht.

Löschen eines Eintrags aus einem B+ Baum

- Beginne an der Wurzel, finde Blatt *L,* wohin der Eintrag gehört..
- Lösche den Eintrag.
  - Wenn *L* mindestens noch halbvoll ist, *fertig!*
  - Wenn *L* nur **d-1** Einträge hat,
    - Versuche umzuverteilen, Borgen vom "Bruder"-Knoten (Nachbarknoten mit dem gleichen Vater wie *L*).
    - Falls Umverteilung scheitert, *mische L* und den Bruder-Knoten.
- Nach diesem Mischen muß der Eintrag aus dem Vater von *L* gelöscht werden (der auf *L* oder dessen Bruder verweist).
- Mischen kann bis zur Wurzel propagiert werden, reduziert somit die Höhe des Baumes.

Beispiel-Baum nach dem Löschen

(vorher Insert 8*), dann Delete 19* und 20*



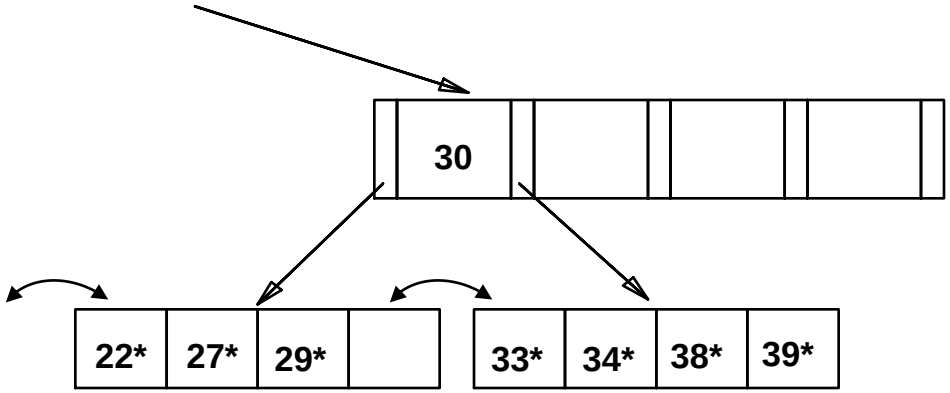
- Löschen 19* ist einfach.
- Löschen 20* erfolgt mit Umverteilung, da Unterlauf im Knoten entstanden ist (Belegung pro Knoten mindestens 2)

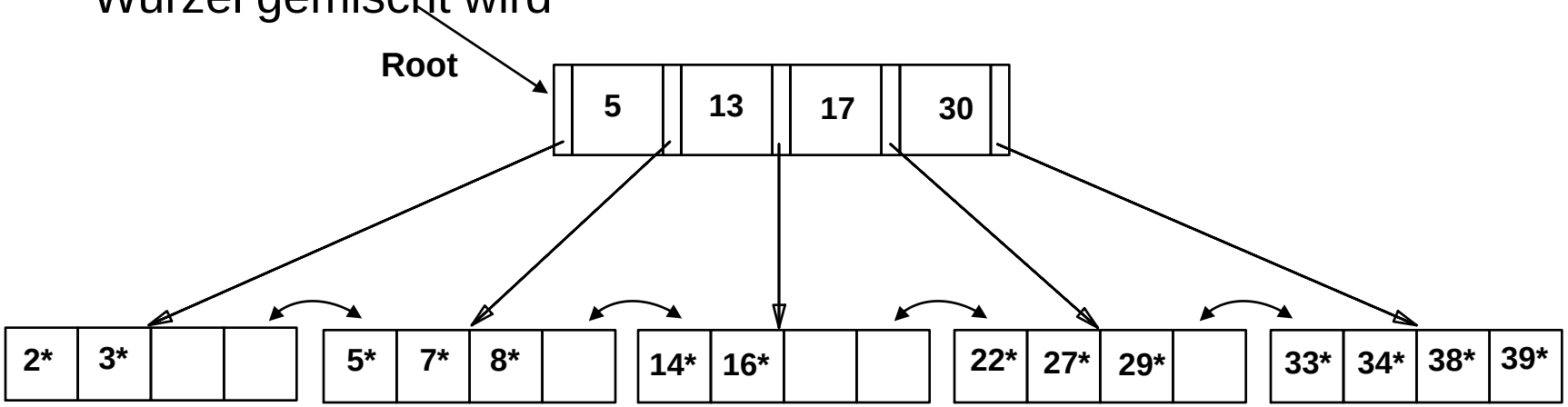
  Mittlerer Schlüssel wird *hochkopiert*.

Weiteres Löschen

... Delete 24*

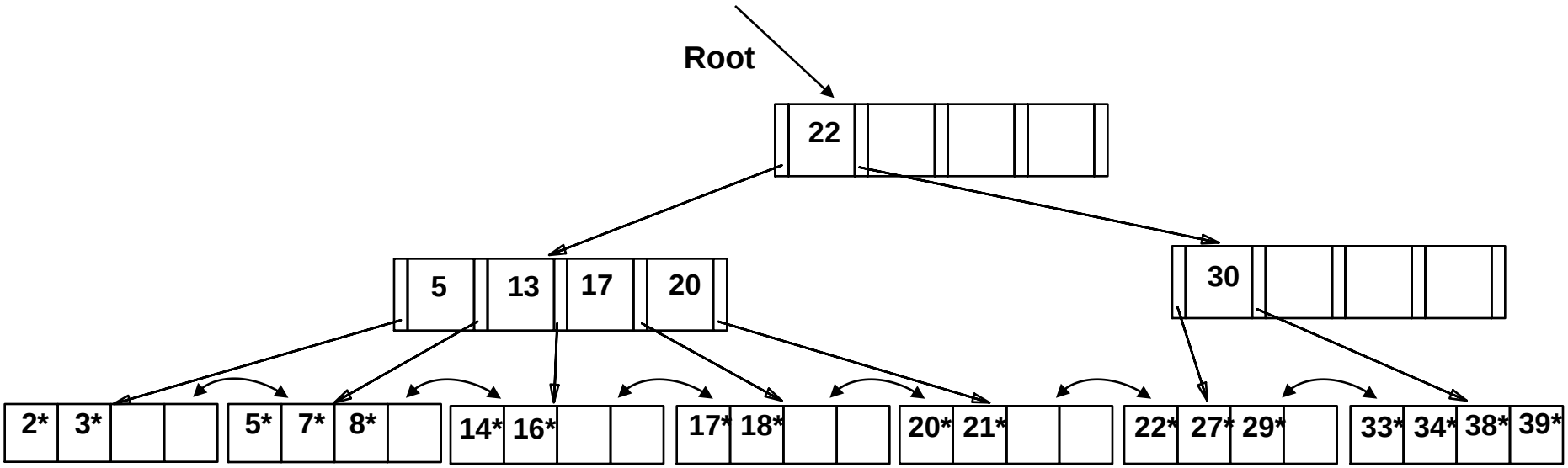
- Mischen erforderlich.
- Zunächst Herüberziehen benachbarter Indexeinträge und Löschen des Verweises im Vater
- dadurch Unterlauf im Vaterknoten, der mit Bruder und Wurzel gemischt wird





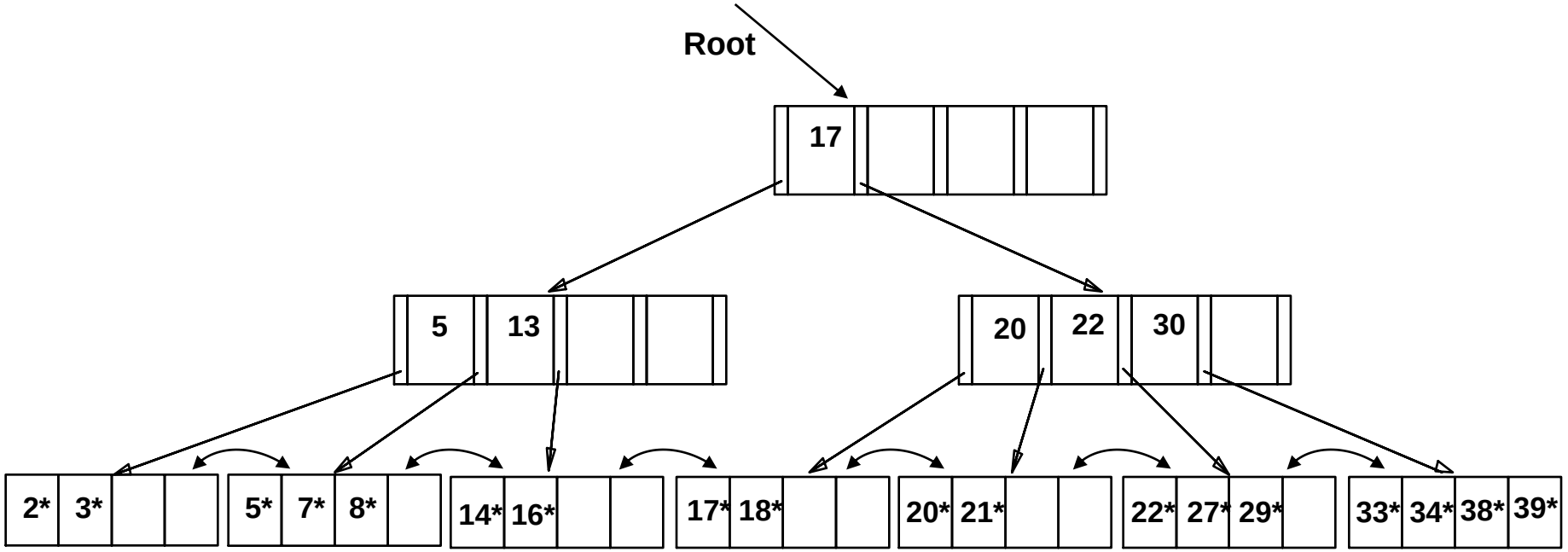
Beispiel einer Umverteilung zwischen Nicht-Blatt-Knoten

- Baum während des Löschens von 24*
- Im Gegensatz zum vorherigen Beispiel, Umverteilung vom linken Sohn der Wurzel zum rechten Sohn (d.h. 30 hat einen Bruder, der Einträge abgeben kann)



Nach der Umverteilung

- Einträge werden umverteilt durch Hochschieben des Split-Eintrages in den Vaterknoten
- Es genügt, den Index-Eintrag mit Schlüssel 20 umzuverteilen; Umverteilung von 17 ist hier nur zur Illustration.



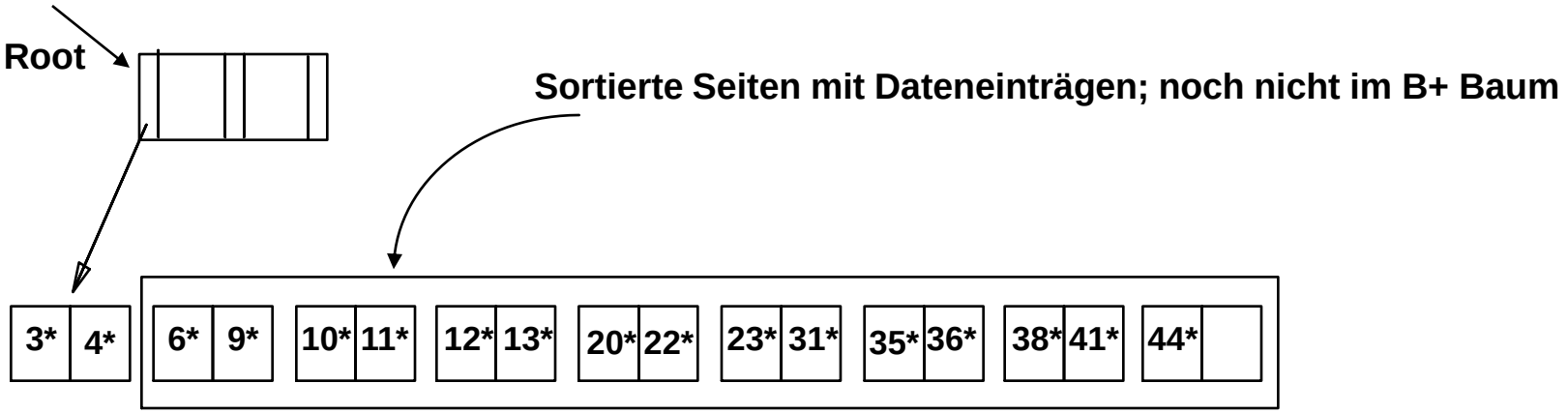
Kompression des Schlüssels

- Höhe eines Baumes abhängig von der Anzahl Dateneinträge und der Größe der Indexeinträge
- Größe der Indexeinträge bestimmt Anzahl der Indexeinträge, die auf eine Seite passen
- Wichtig, um die Anzahl der Sohn-Knoten zu erhöhen
- Optimierungsziel: Erhöhe die Sohn-Anzahl (Fan-Out), minimiere die Baumhöhe (Höhe bestimmt die Anzahl I/Os)
- Alphanumerische Schlüsselwerte in Index-Einträgen lenken nur den Weg; können oft komprimiert werden.
  - Z.B.: Wenn wir benachbarte Index-Einträge haben mit Such-schlüsselwerten *Dannon Yogurt*, *David Smith u*nd *Devarakonda Murthy*, können wir abkürzen: *David Smith* zu *Dav*. (die anderen Schlüssel ebenfalls ...)
    - Korrekt? Nicht unbedingt! Wenn es einen Indexeintrag gäbe *Davey Jones*? (Dann weniger komprimieren, z.B. *David Smith* zu *Davi*)
    - Im allgemeinen gilt: Während der Kompression muß jeder Indexeintrag länger als irgendein Schlüsselwert in seinem linken Teilbaum sein.
- Insert/Delete müssen entsprechend modifiziert werden.

Laden eines B+ Baumes

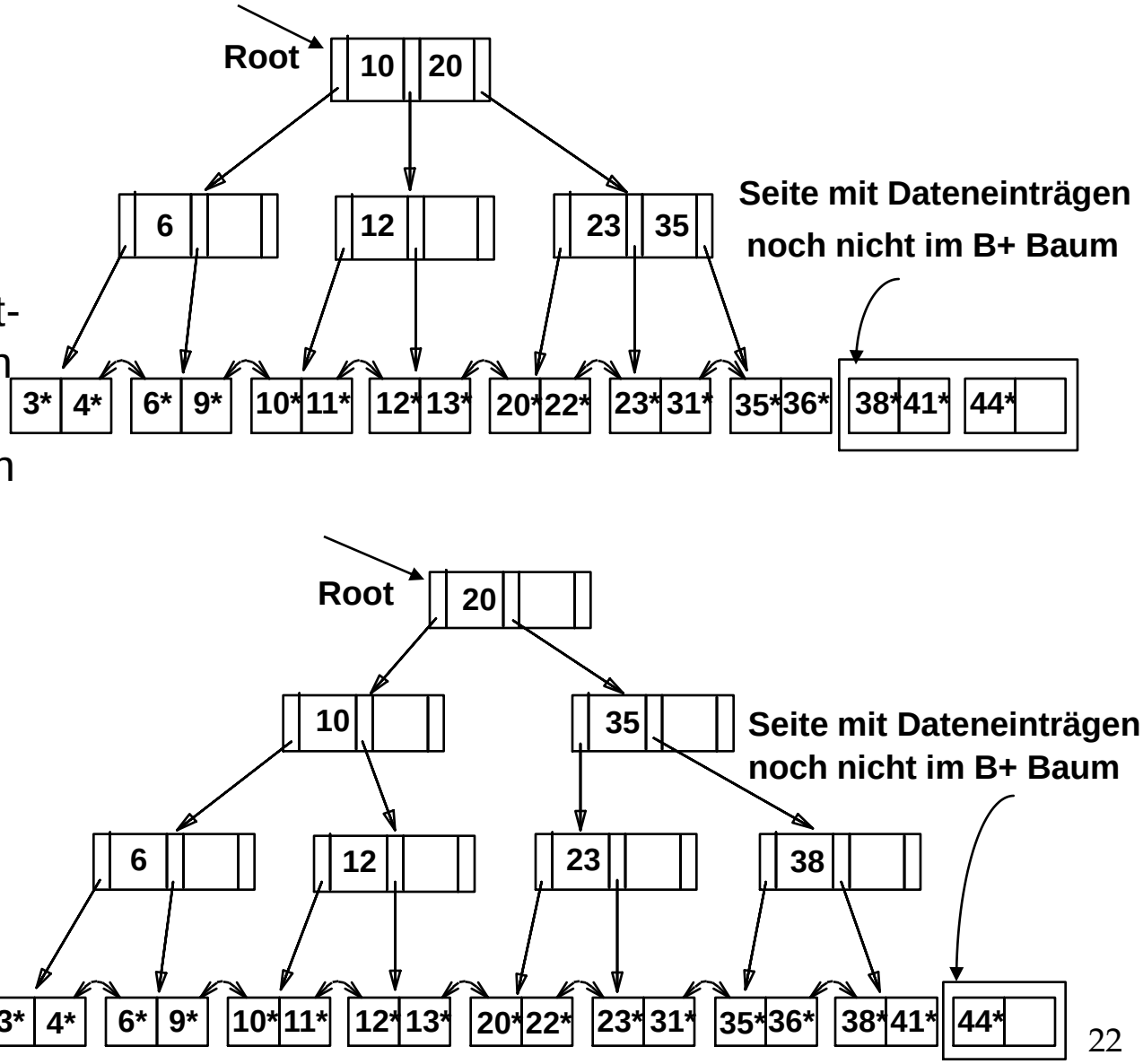
- Wenn wir eine große Anzahl von Sätzen haben, und wir möchten einen B+ Baum auf einem Feld erzeugen, ist dies mit wiederholtem Einfügen von Sätzen sehr langsam.
- *Laden (Bulk Loading)* kann wesentlich effizienter gemacht werden.
- *Initialisierung*: Sortiere alle Dateneinträge, füge Pointer auf die erste (Blatt) Seite in einer neuen (Wurzel) Seite ein.

Beispiel-Baum mit Ordnung d = 1



Laden eines B+ Baumes (Forts.)

- Indexeinträge für Blatt-Seiten immer in die rechteste Indexseite unmittelbar oberhalb Blatt-Ebene eingegeben. Wenn diese gefüllt ist, splitten (Splitten kann den ganzen rechtesten Pfad bis zur Wurzel hochgehen)
- Viel schneller als wiederholte Inserts, besonders wenn Sperren berücksichtigt werden!



Zusammenfassung: Laden

- Option 1: viele Inserts
  - Langsam
  - Keine sequentielle Abspeicherung der Blätter
- Option 2: *Bulk Loading*
  - Hat Vorteile für Concurrency Control (Kontrolle des nebenläufigen Zugriffs auf Seiten)
  - Weniger I/Os beim Aufbau des Baumes
  - Blätter werden sequentiell gespeichert (und auch entsprechend gelinkt)
  - Kontrolle über den “Füll-Faktor” auf den Seiten

Ordnung

- *Ordnung* (**d**) in der Praxis ersetzt durch das Kriterium Platzausnutzung (z.B. Belegung '*mindestens halb-voll*').
  - Knoten in einem B+ Baum entsprechen Seiten
  - Indexseiten (Nicht-Blatt-Knoten) können typischerweise wesentlich mehr Einträge als Blattseiten aufnehmen.
  - Sätze und Suchschlüssel variabler Länge bedeuten: verschiedene Knoten enthalten unterschiedliche Anzahl von Einträgen.
  - Selbst bei Feldern fester Länge können mehrere Sätzen mit dem gleichen Wert des Suchschlüssels (*Duplikate*) zu variabel langen Dateneinträgen führen (bei Verwendung von Alternative 3, d.h. Dateneinträge enthalten Listen von Datensatz-IDs)

Zusammenfassung

- Baum-basierter Index ideal für Bereichsanfragen (Range Query), auch gut für Lookups
- ISAM ist eine statische Struktur:
  - Nur Blattseiten werden modifiziert; benötigt Überlaufseiten
  - Überlaufketten können Performance ruinieren, ausgenommen Datenmenge und Datenverteilung bleiben gleich.
- B+ Baum ist eine dynamische Struktur:
  - Inserts/Deletes lassen den Baum höhen-balanciert: $\log_d N$ Kosten
  - Hohe Ordnung = hohe Anzahl Söhne bedeutet geringe Baumhöhe (ca. 3 oder 4)
  - Fast immer besser als sortierte Datei
  - 67% Belegung des Baues im Durchschnitt
  - Gegenüber ISAM vorzuziehen; paßt sich flexibel ans Wachstum an
  - Wenn Dateneinträge Datensätze sind, können sich Satz-IDs durch Splits ändern!
- Schlüsselkompression erhöht die Ordnung, reduziert Baumhöhe
- Bulk loading wesentlich schneller als wiederholte Inserts, um B+ Bäume für große Datenmengen zu erzeugen
- Verbreitester Index in Datenbank-Management-Systemen wegen seiner Vielseitigkeit. Meist optimierte Komponente eines DBMS.